MS- AI900 certificate learning notes

==================

Learning materials

Trainings: <https://learn.microsoft.com/en-us/credentials/certifications/azure-ai-fundamentals/?practice-assessment-type=certification>

Official documentation: [Azure OpenAI Service documentation - Quickstarts, Tutorials, API Reference - Azure AI services | Microsoft Learn](https://learn.microsoft.com/en-us/azure/ai-services/openai/)

Study guide: [Study guide for Exam AI-900: Microsoft Azure AI Fundamentals | Microsoft Learn](https://learn.microsoft.com/en-us/credentials/certifications/resources/study-guides/ai-900)  
Transformer explainer: [Transformer Explainer: LLM Transformer Model Visually Explained](https://poloclub.github.io/transformer-explainer/)  
  
My Azure machine learning workspace: [Astrid\_AzMachineLearningWorkspace - Microsoft Azure](https://ms.portal.azure.com/#@fdpo.onmicrosoft.com/resource/subscriptions/a84218b0-595f-42a6-9bb6-7e84ad72bd28/resourcegroups/mengjiechai_Machine_Learning_Resource_group/providers/Microsoft.MachineLearningServices/workspaces/Astrid_AzMachineLearningWorkspace/overview)  
  
=====================

Lesson 1 Beginning  
**LLM & SLM**  
LLM: large language models (LLMs). Powerful, more variables, most cost on training and use.

SLM: Small language models. More focused on specific topic areas, and usually cost less.

Common uses of generative AI include:

1. Implementing chatbots and AI agents that assist human users.
2. Creating new documents or other content (often as a starting point for further iterative development)
3. Automated translation of text between languages.
4. Summarizing or explaining complex documents.

**Computer vision** (accomplished by using large numbers of images to train a model)

1. Image classification. A form of computer vision. It trained with images that are labeled with the main subject of the image (in other words, what it's an image of) so that it can analyze unlabeled images and predict the most appropriate label - identifying the subject of the image.



1. Object detection. A more complex form of computer vision. It trained the module to identify the location of specific objects in an image. It provides bounding boxes around the detected objects and labels them with a category (e.g., "car," "dog," "person"). It does not care about the exact pixel-level details of the object. For instance, it will tell you that there’s a car in the image, but it won’t tell you which pixels belong to the car.  
   A banana and an apple next to oranges

   AI-generated content may be incorrect.
2. Semantic segmentation. An advanced form of computer vision. The model can identify the individual pixels in the image that belong to a particular object. It assigns a class label to every pixel in an image, effectively dividing the image into meaningful regions based on the objects or features present. E.g. In the same image, semantic segmentation would color all the pixels corresponding to the car as "car," and the pixels corresponding to the road as "road," providing a detailed mask of the scene.

**Example of Use Cases:**

* **Object Detection**: Self-driving cars need to detect specific objects (cars, pedestrians, traffic signs) to navigate safely.
* **Semantic Segmentation**: Medical imaging (e.g., identifying tumors or organs) requires pixel-level accuracy, which is achieved through semantic segmentation.

Something additional:  
Q: Coding: “For coding, does code will be regarded as Language Model? I mean codes are not images right?”

A: Yes, code can be regarded as a language model—it's a specific subtype of language model that is trained to understand and generate programming code.

Code, being a form of structured text, can be processed by language models that have been trained on it.

AI tools like Copilot, Codex, Tabnine, and CodeBERT are examples of models that have been designed specifically to understand and generate code, treating code just like natural language, but with an emphasis on programming syntax and semantics.

**Speech**

Speech recognition: Convert spoken 🡪 Text. It is one of the tasks of NLP.  
Speech synthesis ( which is known as Text-to-Speech, TTS): Convert Text 🡪 Spoken  
  
  
**Natural Language processing (NLP)**

NLP are based on modules that are trained to do particular type of text analysis. In other words, NLP is a field of AI which is to bridge the gap between human communication (which is naturally unstructured and complex) and machine understanding (which requires structured data).

It is the technology that allows machines to read, interpret, and respond to human language.

**How Does NLP Work?**

NLP involves several key tasks that allow computers to understand and manipulate language, including below 7 tasks.

1. **Text Preprocessing**:
   * Before the computer can process text, it needs to clean and prepare it. This involves:
     + **Tokenization**: Breaking the text into words, phrases, or sentences (called **tokens**).
     + **Stopword Removal**: Removing common words (like "and", "the", "is") that don't add much meaning.
     + **Stemming and Lemmatization**: Reducing words to their base form. For example, “running” becomes “run” (stemming) or “better” becomes “good” (lemmatization).
2. **Part-of-Speech (POS) Tagging**:
   * This task involves identifying the role of each word in a sentence (e.g., noun, verb, adjective). For example, in the sentence "She quickly ran to the store," POS tagging will identify:
     + "She" as a pronoun (noun),
     + "quickly" as an adverb,
     + "ran" as a verb,
     + "store" as a noun.
3. **Named Entity Recognition (NER)**:
   * This task involves identifying and classifying proper nouns (e.g., names of people, places, organizations) in a text.
     + **Example**: In the sentence, "Apple released the iPhone in California," an NLP system might recognize "Apple" as an organization and "California" as a location.
4. **Sentiment Analysis**:
   * This involves determining the **sentiment** or **emotion** expressed in a piece of text—whether it's positive, negative, or neutral.
     + **Example**: For the sentence "I love this phone!" sentiment analysis would classify it as **positive**.
5. **Machine Translation**:
   * Translating text from one language to another.
     + **Example**: Google Translate uses NLP to translate a sentence like "How are you?" from English to Spanish ("¿Cómo estás?").
6. **Speech Recognition**:
   * Converting spoken language into text. This combines NLP with speech recognition technologies to understand spoken words and convert them into written form.
     + **Example**: **Siri** or **Google Assistant** interpreting spoken commands and converting them into text-based actions.
7. **Text Generation**:
   * The AI creates meaningful text from an initial input, like generating creative writing, emails, or news articles. It’s often based on large language models.
     + **Example**: Chatbots (like this one) generate responses based on the context of the conversation.

Examples: Siri, Google Assistant, Customer service chatbots (like on a website).

**Extract data and insights**  
Optical character recognition (OCR) which is a computer vision technology and it is the basis for most document analysis. It can identify the location of text in an image. Therefore, it converts different types of documents—such as scanned paper documents, PDF files, or images captured by a camera—into editable and searchable data.

**Some advanced models of OCR**

**1. Convolutional Neural Networks (CNNs):**

* **CNNs** have been widely used for image processing tasks, and they are especially useful for **character recognition** in OCR. CNNs can recognize and extract features from images at various levels (edges, shapes, textures), making them effective for recognizing printed characters.
* **Example**: CNNs are often used in **Deep Learning-based OCR systems** for **recognizing characters** from images, making them more robust and accurate than traditional template-matching methods.

**2. Recurrent Neural Networks (RNNs):**

* **RNNs**, and specifically **Long Short-Term Memory (LSTM)** networks, are well-suited for **sequence-based tasks** like OCR. They can recognize text from images with varying character sequences and contextual dependencies (e.g., in handwriting recognition).
* **Example**: LSTMs are often used in combination with CNNs to recognize both printed and handwritten text, as they can learn to predict the next characters based on previously recognized text.

**Example OCR Systems** Using RNNs/LSTMs:

* **Tesseract OCR**: Tesseract, an open-source OCR engine, uses LSTM-based models to improve recognition accuracy for both printed and handwritten text.

**Data and insight extraction scenarios**

Common uses of AI to extract data and insights include:

* Automated processing of forms and other documents in a business process - for example, processing an expense claim.
* Large-scale digitization of data from paper forms. For example, scanning and archiving census records.
* Indexing documents for search.
* Identifying key points and follow-up actions from meeting transcripts or recordings.

**Lesson 2 Machine learning**  
https://learn.microsoft.com/en-us/training/modules/fundamentals-machine-learning/

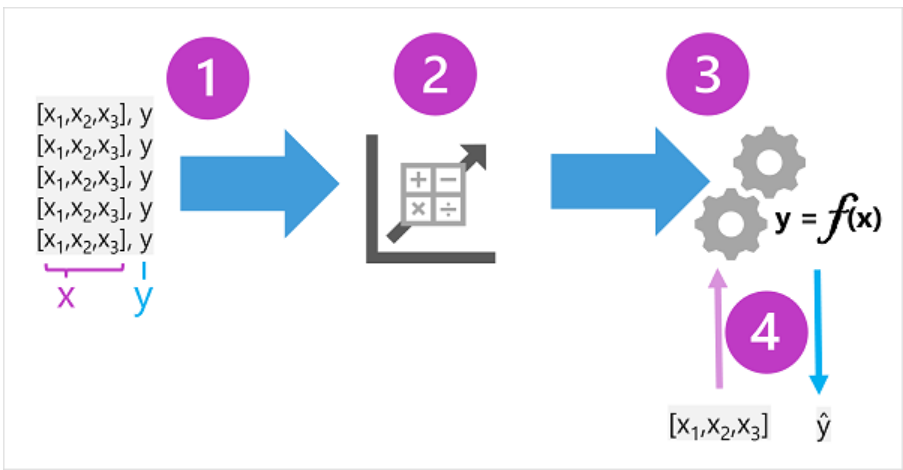
The goal of machine learning: Create a predictive model that can be incorporated into a software application or service.

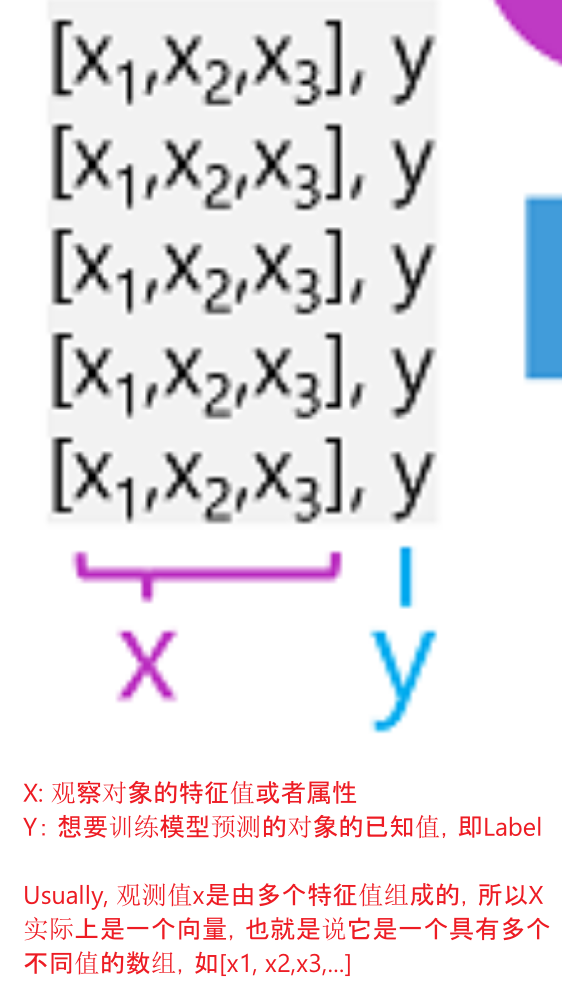
A combination of data science and software engineering.

Data science: Explorer and Prepare data

Software engineering: Using the data above to train/integrate modules into applications, and these applications can be used to predict new data value (inferencing (推理)).

--------------------  
Fundamentally, a machine learning module is:  
One or more input values will be calculated by an encapsulated (封装) function, and the process of defining this function is called ‘Training’. After the function is successfully defined, the process of using it to predict new values called ‘Inferencing’.



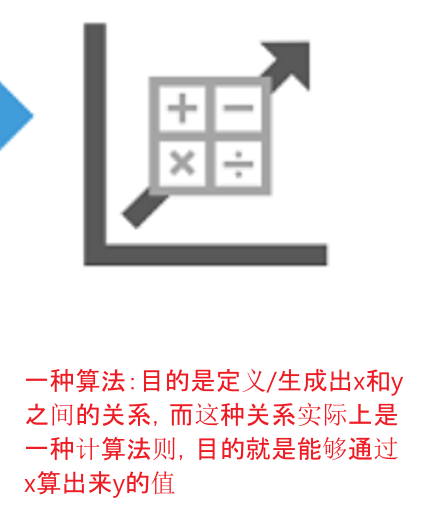
For training data, it actually are consists of past observations, which can be described as below x and y.  
  
Step 1:  
  


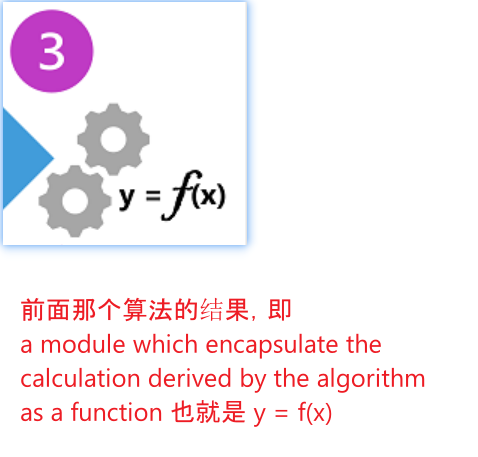
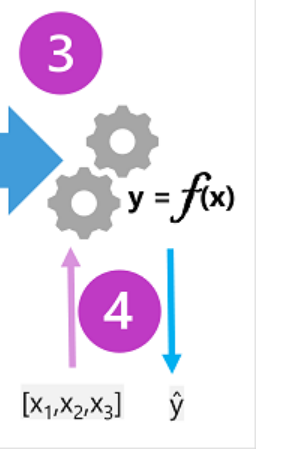
Examples to understand above x & y:  
Example 1:  
The proprietor of an ice cream store might use an app that combines historical sales and weather records to predict how many ice creams they're likely to sell on a given day, based on the weather forecast.

In the ice cream sales scenario, our goal is to train a model that can predict the number of ice cream sales based on the weather. The weather measurements for the day (temperature, rainfall, windspeed, and so on) would be the features (x), and the number of ice creams sold on each day would be the label (y).

Example 2:  
A doctor might use clinical data from past patients to run automated tests that predict whether a new patient is at risk from diabetes based on factors like weight, blood glucose level, and other measurements.

In the medical scenario, the goal is to predict whether or not a patient is at risk of diabetes based on their clinical measurements. The patient's measurements (weight, blood glucose level, and so on) are the features (x), and the likelihood of diabetes (for example, 1 for at risk, 0 for not at risk) is the label (y).

Step 2:   


Step 3:  
  
上一步算法推导出的计算法则被封装成f（x）  
  
Step 4:  
  
Now that the training phase is complete, the trained model can be used for inferencing. The model is essentially a software program that encapsulates the function produced by the training process. You can input a set of feature values, and receive as an output a prediction of the corresponding label. Because the output from the model is a prediction that was calculated by the function, and not an observed value, you'll often see the output from the function shown as ŷ (which is rather delightfully verbalized as "y-hat").

**Part 2: Types of machine learning**  
A diagram of a machine learning

AI-generated content may be incorrect.

Supervised machine learning

On previous part 1, actually we mainly talk about supervised machine learning. x and y  
  
A screenshot of a computer screen

AI-generated content may be incorrect.  
  
Regression (回归)  
A form of machine learning in which label predicted by the model is a numeric value.   
例如：  
根据气温、降雨量和风速，计算出某一天冰淇淋的销量。

根据房产面积（平方英尺）、卧室数量以及所在位置的社会经济指标，计算出房产的售价。

* 典型算法：线性回归、梯度提升树（GBDT）、神经网络。

Classification  
In this form of machine learning, the label represents a categorization or a class. There are two common scenarios:  
  
1. Binary classification  
In binary classification, the label determines whether the observed item is (or isn't) an instance of a specific class. Or put another way, binary classification models predict one of two mutually exclusive outcomes.  
  
例如：

根据体重、年龄、血糖水平等临床指标，判断患者是否有患糖尿病的风险。

根据收入、信用记录、年龄和其他因素，判断银行客户是否会拖欠贷款。

根据人口统计属性和历史购买记录，邮件列表客户是否会对营销优惠做出积极回应。  
  
the model predicts a binary *true*/*false* or *positive/negative* prediction for a single possible class.

2. Multiclass classification  
*Multiclass classification* extends binary classification to predict a label that represents one of multiple possible classes. For example,

根据企鹅的体型，预测其种类（阿德利企鹅、巴布亚企鹅或帽带企鹅）。

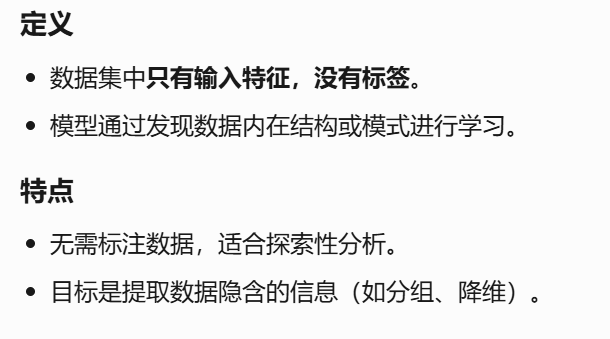
根据电影的演员阵容、导演和预算，预测其类型（喜剧、恐怖、爱情、冒险或科幻）。  
  
In most scenarios that involve a known set of multiple classes, multiclass classification is used to predict mutually exclusive labels. For example, a penguin can't be both a *Gentoo* and an *Adelie*. However, there are also some algorithms that you can use to train *multilabel* classification models, in which there may be more than one valid label for a single observation. For example, a movie could potentially be categorized as both *science fiction* and *comedy*.

| **​​特征​​** | **​​二分类（Binary Classification）​​** | **​​多分类（Multiclass Classification）​​** |
| --- | --- | --- |
| ​**​目标类别数​**​ | 仅预测​**​两个互斥类别​**​（如“是/否”“0/1”）。 | 预测​**​三个或更多互斥类别​**​（如“猫/狗/鸟”“A/B/C/D”）。 |
| ​**​输出形式​**​ | 概率值（0~1）或二元标签。 | 多个类别的概率分布或单一标签。 |
| ​**​算法适配性​**​ | 部分算法原生支持（如逻辑回归、SVM）。 | 需扩展或调整算法（如Softmax回归、决策树）。 |

（学习过程中发现，对于supervised machine learning ,数据集的拆分是很重要的，无论是regression, binary classification还是multi-class classification都会涉及数据集的拆分，通常拆分为训练集、验证集、测试集）  
  
**Unsupervised machine learning**

*Unsupervised* machine learning involves training models using data that consists only of *feature* values without any known labels. Unsupervised machine learning algorithms determine relationships between the features of the observations in the training data.

Only feature x no label y.

  
  
Clustering  
The most common form of unsupervised machine learning is *clustering*. A clustering algorithm identifies similarities between observations based on their features, and groups them into discrete (离散的) clusters.

例如：

根据花朵的大小、叶子数量和花瓣数量，对相似的花朵进行分类。

根据人口统计属性和购买行为，识别相似的客户群体。

In some ways, clustering is similar to multiclass classification; in that it categorizes observations into discrete groups. The difference is that when using classification, you already know the classes to which the observations in the training data belong; so the algorithm works by determining the relationship between the features and the known classification label. In clustering, there's no previously known cluster label and the algorithm groups the data observations based purely on similarity of features.

In some cases, clustering is used to determine the set of classes that exist before training a classification model. For example, you might use clustering to segment your customers into groups, and then analyze those groups to identify and categorize different classes of customer (*high value - low volume*, *frequent small purchaser*, and so on). You could then use your categorizations to label the observations in your clustering results and use the labeled data to train a classification model that predicts to which customer category a new customer might belong.

**Part 3 Regression (回归)**

A diagram of a diagram

AI-generated content may be incorrect.

1. Split the training data (randomly) to create a dataset with which to train the model while holding back a subset of the data that you'll use to validate the trained model.

拆分训练数据（随机）以创建一个数据集，用于训练模型，同时保留用于验证训练模型的数据子集。

1. Use an algorithm to fit the training data to a model. In the case of a regression model, use a regression algorithm such as *linear regression*.

使用算法将训练数据拟合到模型中。如果是回归模型，请使用回归算法，例如线性回归。

1. Use the validation data you held back to test the model by predicting labels for the features.

使用保留的验证数据，通过预测特征的标签来测试模型。

1. Compare the known *actual* labels in the validation dataset to the labels that the model predicted. Then aggregate (汇总) the differences between the *predicted* and *actual* label values to calculate a metric that indicates how accurately the model predicted for the validation data.

将验证数据集中已知的实际标签与模型预测的标签进行比较。然后，汇总预测标签值和实际标签值之间的差异，以计算一个指标，该指标指示模型对验证数据的预测准确度。  
  
After each train, validate, and evaluate iteration, you can repeat the process with different algorithms and parameters until an acceptable evaluation metric is achieved.

**Regression evaluation metrics**

Based on the differences between the predicted and actual values, you can calculate some common metrics that are used to evaluate a regression model.

**Mean Absolute Error (MAE)**

The variance in this example indicates by how many ice creams each prediction was wrong. It doesn't matter if the prediction was *over* or *under* the actual value (so for example, -3 and +3 both indicate a variance of 3). This metric is known as the *absolute error* for each prediction, and can be summarized for the whole validation set as the **mean absolute error** (MAE).

In the ice cream example, the mean (average) of the absolute errors (2, 3, 3, 1, 2, and 3) is **2.33**.

**Mean Squared Error (MSE)**

The mean absolute error metric takes all discrepancies between predicted and actual labels into account equally. However, it may be more desirable to have a model that is consistently wrong by a small amount than one that makes fewer, but larger errors. One way to produce a metric that "amplifies" larger errors by *squaring* the individual errors and calculating the mean of the squared values. This metric is known as the **mean squared error** (MSE).

In our ice cream example, the mean of the squared absolute values (which are 4, 9, 9, 1, 4, and 9) is **6**.

**Root Mean Squared Error (RMSE)**

The mean squared error helps take the magnitude of errors into account, but because it *squares* the error values, the resulting metric no longer represents the quantity measured by the label. In other words, we can say that the MSE of our model is 6, but that doesn't measure its accuracy in terms of the number of ice creams that were mispredicted; 6 is just a numeric score that indicates the level of error in the validation predictions.

If we want to measure the error in terms of the number of ice creams, we need to calculate the *square root* of the MSE; which produces a metric called, unsurprisingly, **Root Mean Squared Error**. In this case √6, which is **2.45** (ice creams).

**Coefficient of determination (R2)**

All of the metrics so far compare the discrepancy between the predicted and actual values in order to evaluate the model. However, in reality, there's some natural random variance in the daily sales of ice cream that the model takes into account. In a linear regression model, the training algorithm fits a straight line that minimizes the mean variance between the function and the known label values. The **coefficient of determination** (more commonly referred to as **R2** or **R-Squared**) is a metric that measures the proportion of variance in the validation results that can be explained by the model, as opposed to some anomalous aspect of the validation data (for example, a day with a highly unusual number of ice creams sales because of a local festival).

The calculation for R2 is more complex than for the previous metrics. It compares the sum of squared differences between predicted and actual labels with the sum of squared differences between the actual label values and the mean of actual label values, like this:

***R2 = 1- ∑(y-ŷ)2 ÷ ∑(y-ȳ)2***

Don't worry too much if that looks complicated; most machine learning tools can calculate the metric for you. The important point is that the result is a value between 0 and 1 that describes the proportion of variance explained by the model. In simple terms, the closer to 1 this value is, the better the model is fitting the validation data. In the case of the ice cream regression model, the R2 calculated from the validation data is **0.95**.

**Part 4 Binary classification**

Completed100 XP

* 12 minutes

Classification, like regression, is a *supervised* machine learning technique; and therefore follows the same iterative process of training, validating, and evaluating models. Instead of calculating numeric values like a regression model, the algorithms used to train classification models calculate *probability* values for class assignment and the evaluation metrics used to assess model performance compare the predicted classes to the actual classes.

*Binary classification* algorithms are used to train a model that predicts one of two possible labels for a single class. Essentially, predicting ***true*** or ***false***. In most real scenarios, the data observations used to train and validate the model consist of multiple feature (***x***) values and a ***y*** value that is either **1** or **0**.

**Example - binary classification**

To understand how binary classification works, let's look at a simplified example that uses a single feature (***x***) to predict whether the label ***y*** is 1 or 0. In this example, we'll use the blood glucose level of a patient to predict whether or not the patient has diabetes. Here's the data with which we'll train the model:

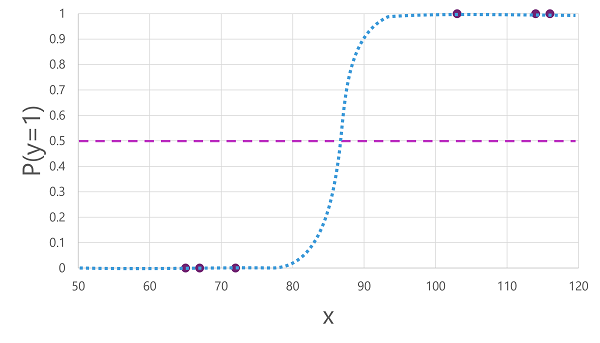
Expand table

| **Diagram of a syringe.** | **Diagram of a diabetic and non-diabetic person.** |
| --- | --- |
| **Blood glucose (x)** | **Diabetic? (y)** |
| 67 | 0 |
| 103 | 1 |
| 114 | 1 |
| 72 | 0 |
| 116 | 1 |
| 65 | 0 |

**Training a binary classification model**

To train the model, we'll use an algorithm to fit the training data to a function that calculates the *probability* of the class label being *true* (in other words, that the patient has diabetes). Probability is measured as a value between 0.0 and 1.0, such that the *total* probability for *all* possible classes is 1.0. So for example, if the probability of a patient having diabetes is 0.7, then there's a corresponding probability of 0.3 that the patient isn't diabetic.

There are many algorithms that can be used for binary classification, such as *logistic regression*, which derives a *sigmoid* (S-shaped) function with values between 0.0 and 1.0, like this:



**Note**

Despite its name, in machine learning *logistic regression* is used for classification, not regression. The important point is the *logistic* nature of the function it produces, which describes an S-shaped curve between a lower and upper value (0.0 and 1.0 when used for binary classification).

The function produced by the algorithm describes the probability of ***y*** being true (*y*=1) for a given value of ***x***. Mathematically, you can express the function like this:

***f(x) = P(y=1 | x)***

For three of the six observations in the training data, we know that ***y*** is definitely *true*, so the probability for those observations that *y*=1 is **1.0** and for the other three, we know that ***y*** is definitely *false*, so the probability that *y*=1 is **0.0**. The S-shaped curve describes the probability distribution so that plotting a value of ***x*** on the line identifies the corresponding probability that ***y*** is **1**.

The diagram also includes a horizontal line to indicate the *threshold* at which a model based on this function will predict *true* (**1**) or *false* (**0**). The threshold lies at the mid-point for ***y*** (*P(y) = 0.5*). For any values at this point or above, the model will predict *true* (**1**); while for any values below this point it will predict *false* (**0**). For example, for a patient with a blood glucose level of 90, the function would result in a probability value of 0.9. Since 0.9 is higher than the threshold of 0.5, the model would predict *true* (**1**) - in other words, the patient is predicted to have diabetes.

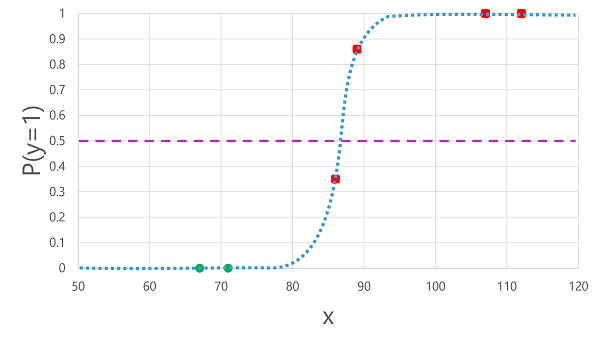
**Evaluating a binary classification model**

As with regression, when training a binary classification model you hold back a random subset of data with which to validate the trained model. Let's assume we held back the following data to validate our diabetes classifier:

Expand table

| **Blood glucose (x)** | **Diabetic? (y)** |
| --- | --- |
| 66 | 0 |
| 107 | 1 |
| 112 | 1 |
| 71 | 0 |
| 87 | 1 |
| 89 | 1 |

Applying the logistic function we derived previously to the ***x*** values results in the following plot.



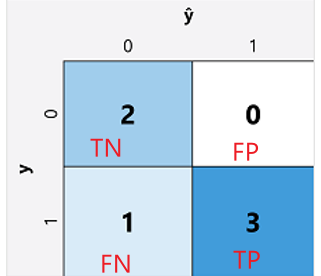
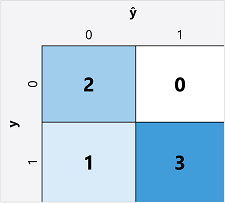
Based on whether the probability calculated by the function is above or below the threshold, the model generates a predicted label of 1 or 0 for each observation. We can then compare the *predicted* class labels (***ŷ***) to the *actual* class labels (***y***), as shown here:

Expand table

| **Blood glucose (x)** | **Actual diabetes diagnosis (y)** | **Predicted diabetes diagnosis (ŷ)** |
| --- | --- | --- |
| 66 | 0 | 0 |
| 107 | 1 | 1 |
| 112 | 1 | 1 |
| 71 | 0 | 0 |
| 87 | 1 | 0 |
| 89 | 1 | 1 |

**Binary classification evaluation metrics**

The first step in calculating evaluation metrics for a binary classification model is usually to create a matrix of the number of correct and incorrect predictions for each possible class label:



This visualization is called a *confusion matrix*, and it shows the prediction totals where:

* ŷ=0 and y=0: *True negatives* (TN)
* ŷ=1 and y=0: *False positives* (FP)
* ŷ=0 and y=1: *False negatives* (FN)
* ŷ=1 and y=1: *True positives* (TP)

The arrangement of the confusion matrix is such that correct (*true*) predictions are shown in a diagonal line from top-left to bottom-right. Often, color-intensity is used to indicate the number of predictions in each cell, so a quick glance at a model that predicts well should reveal a deeply shaded diagonal trend.

**Accuracy**

The simplest metric you can calculate from the confusion matrix is *accuracy* - the proportion of predictions that the model got right. Accuracy is calculated as:

***(TN+TP) ÷ (TN+FN+FP+TP)***

In the case of our diabetes example, the calculation is:

(2+3) ÷ (2+1+0+3)

= 5 ÷ 6

= **0.83**

So for our validation data, the diabetes classification model produced correct predictions 83% of the time.

Accuracy might initially seem like a good metric to evaluate a model, but consider this. Suppose 11% of the population has diabetes. You could create a model that always predicts **0**, and it would achieve an accuracy of 89%, even though it makes no real attempt to differentiate between patients by evaluating their features. What we really need is a deeper understanding of how the model performs at predicting **1** for positive cases and **0** for negative cases.

**Recall**

*Recall* is a metric that measures the proportion of positive cases that the model identified correctly. In other words, compared to the number of patients who *have* diabetes, how many did the model *predict* to have diabetes?  
召回率 (Recall) 是衡量模型正确识别的阳性病例比例的指标。换句话说，与实际患有糖尿病的患者数量相比，模型预测有多少人患有糖尿病？

The formula for recall is:

***TP ÷ (TP+FN)***

For our diabetes example:

3 ÷ (3+1)

= 3 ÷ 4

= **0.75**

So our model correctly identified 75% of patients who have diabetes as having diabetes.

**Precision**

*Precision* is a similar metric to recall, but measures the proportion of predicted positive cases where the true label is actually positive. In other words, what proportion of the patients *predicted* by the model to have diabetes actually *have* diabetes?  
精确率 (Precision) 与召回率类似，但衡量的是预测阳性病例中实际标签为阳性的病例比例。换句话说，模型预测患有糖尿病的患者中，有多少比例实际患有糖尿病？

The formula for precision is:

***TP ÷ (TP+FP)***

For our diabetes example:

3 ÷ (3+0)

= 3 ÷ 3

= **1.0**

So 100% of the patients predicted by our model to have diabetes do in fact have diabetes.

**F1-score**

*F1-score* is an overall metric that combined recall and precision. The formula for F1-score is:

***(2 x Precision x Recall) ÷ (Precision + Recall)***

For our diabetes example:

(2 x 1.0 x 0.75) ÷ (1.0 + 0.75)

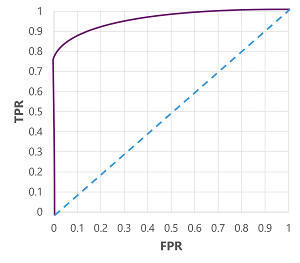
= 1.5 ÷ 1.75

**= 0.86**

**Area Under the Curve (AUC)**

Another name for recall is the *true positive rate* (TPR), and there's an equivalent metric called the *false positive rate* (FPR) that is calculated as **FP÷(FP+TN)**. We already know that the TPR for our model when using a threshold of 0.5 is 0.75, and we can use the formula for FPR to calculate a value of 0÷2 = 0.

Of course, if we were to change the threshold above which the model predicts *true* (**1**), it would affect the number of positive and negative predictions; and therefore change the TPR and FPR metrics. These metrics are often used to evaluate a model by plotting a *received operator characteristic* (ROC) curve that compares the TPR and FPR for every possible threshold value between 0.0 and 1.0:



The ROC curve for a perfect model would go straight up the TPR axis on the left and then across the FPR axis at the top. Since the plot area for the curve measures 1x1, the area under this perfect curve would be 1.0 (meaning that the model is correct 100% of the time). In contrast, a diagonal line from the bottom-left to the top-right represents the results that would be achieved by randomly guessing a binary label; producing an area under the curve of 0.5. In other words, given two possible class labels, you could reasonably expect to guess correctly 50% of the time.

In the case of our diabetes model, the curve above is produced, and the **area under the curve** (AUC) metric is 0.875. Since the AUC is higher than 0.5, we can conclude the model performs better at predicting whether or not a patient has diabetes than randomly guessing.

**Part 5 Multiclass classification**

*Multiclass classification* is used to predict to which of multiple possible classes an observation belongs. As a supervised machine learning technique, it follows the same iterative *train, validate, and evaluate* process as regression and binary classification in which a subset of the training data is held back to validate the trained model.

**Example - multiclass classification**

Multiclass classification algorithms are used to calculate probability values for multiple class labels, enabling a model to predict the *most probable* class for a given observation.

Let's explore an example in which we have some observations of penguins, in which the flipper length (***x***) of each penguin is recorded. For each observation, the data includes the penguin species (***y***), which is encoded as follows:

* 0: Adelie
* 1: Gentoo
* 2: Chinstrap

**Note**

As with previous examples in this module, a real scenario would include multiple feature (***x***) values. We'll use a single feature to keep things simple.

Expand table

| **Diagram of a measuring ruler.** | **Diagram of three penguins.** |
| --- | --- |
| **Flipper length (x)** | **Species (y)** |
| 167 | 0 |
| 172 | 0 |
| 225 | 2 |
| 197 | 1 |
| 189 | 1 |
| 232 | 2 |
| 158 | 0 |

**Training a multiclass classification model**

To train a multiclass classification model, we need to use an algorithm to fit the training data to a function that calculates a probability value for each possible class. There are two kinds of algorithm you can use to do this:

* One-vs-Rest (OvR) algorithms
* Multinomial algorithms

**One-vs-Rest (OvR) algorithms**

One-vs-Rest algorithms train a binary classification function for each class, each calculating the probability that the observation is an example of the target class. Each function calculates the probability of the observation being a specific class compared to *any* other class. For our penguin species classification model, the algorithm would essentially create three binary classification functions:

* ***f0(x) = P(y=0 | x)***
* ***f1(x) = P(y=1 | x)***
* ***f2(x) = P(y=2 | x)***

Each algorithm produces a sigmoid function that calculates a probability value between 0.0 and 1.0. A model trained using this kind of algorithm predicts the class for the function that produces the highest probability output.

**Multinomial algorithms**

As an alternative approach is to use a multinomial algorithm, which creates a single function that returns a multi-valued output. The output is a *vector* (an array of values) that contains the *probability distribution* for all possible classes - with a probability score for each class which when totaled add up to 1.0:

***f(x) =[P(y=0|x), P(y=1|x), P(y=2|x)]***

An example of this kind of function is a *softmax* function, which could produce an output like the following example:

[0.2, 0.3, 0.5]

The elements in the vector represent the probabilities for classes 0, 1, and 2 respectively; so in this case, the class with the highest probability is **2**.

Regardless of which type of algorithm is used, the model uses the resulting function to determine the most probable class for a given set of features (***x***) and predicts the corresponding class label (***y***).

**Evaluating a multiclass classification model**

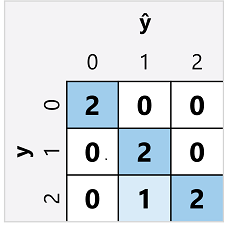
You can evaluate a multiclass classifier by calculating binary classification metrics for each individual class. Alternatively, you can calculate aggregate metrics that take all classes into account.

Let's assume that we've validated our multiclass classifier, and obtained the following results:

Expand table

| **Flipper length (x)** | **Actual species (y)** | **Predicted species (ŷ)** |
| --- | --- | --- |
| 165 | 0 | 0 |
| 171 | 0 | 0 |
| 205 | 2 | 1 |
| 195 | 1 | 1 |
| 183 | 1 | 1 |
| 221 | 2 | 2 |
| 214 | 2 | 2 |

The confusion matrix for a multiclass classifier is similar to that of a binary classifier, except that it shows the number of predictions for each combination of *predicted* (***ŷ***) and *actual* class labels (***y***):



From this confusion matrix, we can determine the metrics for each individual class as follows:

Expand table

| **Class** | **TP** | **TN** | **FP** | **FN** | **Accuracy** | **Recall** | **Precision** | **F1-Score** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 2 | 5 | 0 | 0 | 1.0 | 1.0 | 1.0 | 1.0 |
| **1** | 2 | 4 | 1 | 0 | 0.86 | 1.0 | 0.67 | 0.8 |
| **2** | 2 | 4 | 0 | 1 | 0.86 | 0.67 | 1.0 | 0.8 |

To calculate the overall accuracy, recall, and precision metrics, you use the total of the *TP*, *TN*, *FP*, and *FN* metrics:

* **Overall accuracy** = (13+6)÷(13+6+1+1) = **0.90**
* **Overall recall** = 6÷(6+1) = **0.86**
* **Overall precision** = 6÷(6+1) = **0.86**

The overall F1-score is calculated using the overall recall and precision metrics:

* **Overall F1-score** = (2x0.86x0.86)÷(0.86+0.86) = **0.86**

Q: Please tell me what are **face detection, facial analysis, facial recognition**? And what are differences between them

A:

**😊 1. Face Detection**

* **What it is**: Finding faces in an image or video.
* **What it does**: Draws a **box** around each face.
* **Goal**: Just to say, *“Hey, there’s a face here.”*

✅ Example:  
You upload a group photo → it returns 5 bounding boxes over 5 faces.

🧠 Tech: Computer Vision (no need to know whose face it is)

**🧠 2. Facial Analysis**

* **What it is**: Understanding **features or expressions** of a face.
* **What it does**:
  + Detects **age**, **gender**, **emotion** (happy, sad, angry)
  + Identifies things like if the person is wearing **glasses**, **beard**, etc.

✅ Example:  
“Female, approx. 28 years old, smiling”

🧠 Tech: Combines Computer Vision + some ML/NLP models (for interpreting expressions)

**🧑‍🦰 3. Facial Recognition**

* **What it is**: Identifying or verifying **who** a person is.
* **What it does**:
  + Matches a face against a database of known faces.
  + Can be used for **authentication** or **searching**.

✅ Example:  
“This face matches John Doe in the employee database.”

🧠 Tech: Computer Vision + deep learning embeddings + matching algorithms

| **Task** | **Purpose** | **Output** | **Used For** |
| --- | --- | --- | --- |
| **Face Detection** | Find faces in images | Bounding boxes around faces | Cameras, filters |
| **Facial Analysis** | Understand facial features | Age, gender, emotion, etc. | Demographics, ads |
| **Facial Recognition** | Identify the person | Person’s name or ID | Security, logins |